加微信：642945106 发送“ 赠送 ”领取赠送精品课程
发数字“ 2 ”获取众筹列表

01 | 程序员为什么要关心代码性能?

2019-12-02 庄振运

讲述：庄振运
时长 13:47　大小 12.64M

你好，我是庄振运。

感谢你加入这个专栏的学习，我也非常高兴能有机会和你一起探索这个领域。

我在计算机和互联网行业已经研究和工作近 20 年了，一直从事性能优化和容量管理相关的工作。从今天起，我就和你分享我这些年的经验和感悟。

提起计算机和互联网，多数人首先想到的职业是程序员。中国有多少程序员呢？很多人估计有 600 万左右。全球的人数就更多了，肯定超过 2000 万。

我虽然也在互联网领域，也做过几年写程序的工作，但是现在的工作，严格意义上不算程序员，而是性能工程师。不过我和很多程序员朋友一起工作过，也讨论过。谈到性能优化和系统容量管理的时候，一开始他们经常会问我一个问题，就是程序员为什么需要了解性能和容量这些东西？通俗点说，这个问题就是：我就是一介程序员，性能和系统容量听起来很重要，但与我何干？

这个问题问得很好。我可以和你肯定地说，程序员应该关心，也必须关心代码性能和系统容量。今天这一讲，我们先说说程序员为什么需要关心性能。

怎么定义“性能”和“性能好”？

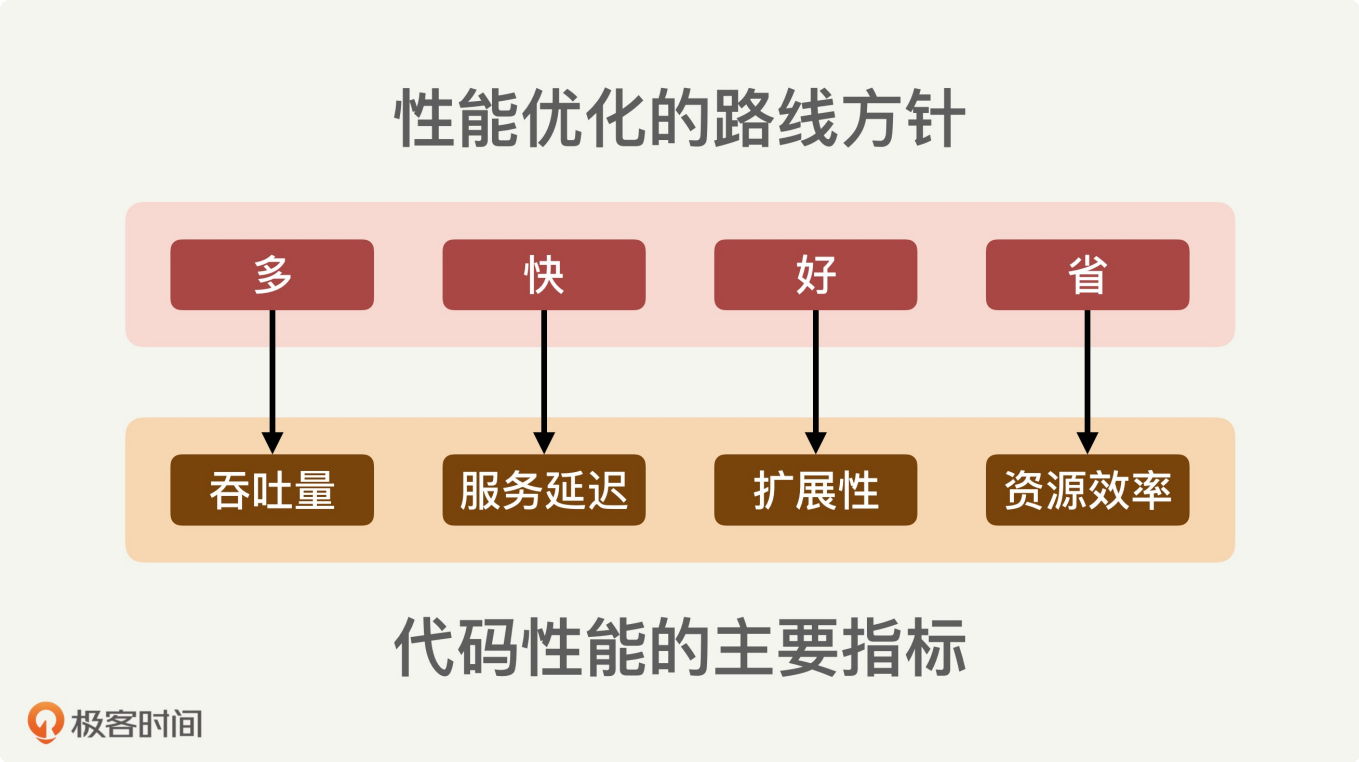
说起代码性能，首先我们需要弄清楚什么样的代码算是性能好？怎么样算是性能不好？

代码性能表现在很多方面和指标，比较常见的几个指标有吞吐量（Throughput）、服务延迟（Service latency）、扩展性（Scalability）和资源使用效率（Resource Utilization）。

- 吞吐量：单位时间处理请求的数量。
- 服务延迟：客户请求的处理时间。
- 扩展性：系统在高压的情况下能不能正常处理请求。
- 资源使用效率：单位请求处理所需要的资源量（比如 CPU，内存等）。

必须说明的是，这几个指标之外，根据场景，还可以有其他性能指标，比如可靠性（Reliability）。可靠性注重的是在极端情况下能不能持续处理正常的服务请求。不过，我们这个专栏的讨论，主要围绕前四个更常见的目标。

性能好的代码，可以用四个字来概括：“多快好省”。



看到这四个字，你可能想起了咱们国家当年制定的大跃进总路线，那就是：“鼓足干劲、力争上游、多快好省地建设社会主义”。没错，高性能代码的要求和这个“社会主义建设总路线”相当一致。这里的“多”，就是吞吐量大；“快”，就是服务延迟低；“好”，就是扩展性好；“省”，就是资源使用量低（也即是资源使用效率高）。

用这样的四个指标来衡量，那么性能不好的代码的表现就是：吞吐量小、延迟大、扩展性差、资源使用高（资源使用效率低）。

程序员为什么要关心代码性能？

对程序员来讲，写出的代码就是他的产品、他的生命线、他的形象和价值。代码性能不好，就是质量差，不靠谱。轻者影响程序员的声誉，重者影响他的工作。

对一个公司来讲，产品质量差，公司或许会倒闭。对程序员所在的互联网公司而言，如果公司的业务依赖于程序员写的代码，那么代码性能差，关键时刻掉链子，比如双十一促销的时候，公司的业务性能就会经常出问题，进而会影响公司的运营和营收，这可是天大的事情。

因此，如果一个程序员写出性能很差的代码，无异于耍流氓，并且相关程序员的工作也很难保住。

防止断更 请务必加
首发微信：17161495665

反过来讲，如果写出的代码性能很高，那代码的作者必定是我们大家认可的“靠谱”程序员，少不了“人见人爱”——客户喜欢，同事喜欢，领导也喜欢。

不同级别的程序员都需要关心性能

还有些朋友或许认为：代码性能是**某些人或者其他人**应该负责的；我就负责把代码写出来，优化的事，他们负责。这里的“某些人和其他人”可以是指软件测试人员、运维人员、技术专家，或者是性能工程师。

这种想法也是不对的。我下面就用几个案例来举例说明，代码性能是各个级别的程序员都应该关心和负责的。事实上，程序员从学校出来开始，一步步地在职业上攀升，每一步都应该和性能结伴而行。

我用一张图来表示一个成功程序员的技术职业轨迹（注意里面的职位和年限仅供参考）。

学生刚刚从学校毕业，加入互联网公司，一般是入门级程序员。工作 1 到 3 年后，就成为普通的程序员。工作三五年后，可以算是资深程序员。工作 6 到 10 年后，可以成长为技术专家。10 年以上，可能成为高级专家或者架构师。

举例 1：刚入门的程序员

小李刚刚大学毕业，进入一个互联网公司。

领导给他的任务是写一个小模块，其中有一个需求是统计两个日期之间有几个正常工作日（也就是多少是周一到周五）。小李采取的是简单暴力法，就是用一个循环，循环的起始和截至日期就是给定的两个日期。在循环里面，对每一个日期判定一次，确定是工作日还是休息日，然后把工作日累加起来。

这样的代码显然性能不高，生产环境里面跑起来很快就会出问题。比如，如果两个日期差距很大，这个模块可能就需要很长时间才能处理完。

如果小李注重代码性能，他完全可以用更高效的方法，比如快速判定给定的两个日期间有多少个星期，然后乘以 5，因为每个星期有 5 个工作日。然后，对头尾的星期进行特殊处理。这样的代码跑起来快多了。我可以想象，小李在优化完代码后，或许会吟诵两句“何当金络脑，快走踏清秋”来形容新代码的性能。

举例 2：普通的程序员

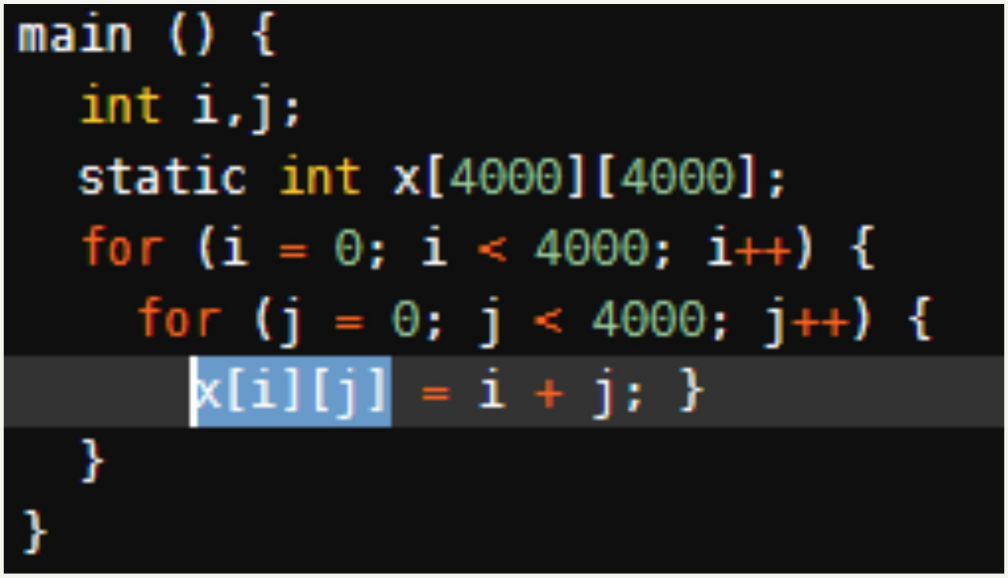
小王做程序员 2 年了，在公司里已经可以独立负责一个模块了。有一天，他需要把一个二维整数数组进行重新赋值，于是，他写出了下面的二重循环：

拼课微信：17161143665

```
main () {
  int i,j;
  static int x[4000][4000];
  for (i = 0; i < 4000; i++) {
    for (j = 0; j < 4000; j++) {
      x[j][i] = i + j; }
  }
}
```

极客时间

如果小王了解计算机内存和缓存的知识以及大小，他或许会写出下面的循环。虽然只有两个字母的差别，性能却提升了很多倍。



极客时间

原因是什么呢?

因为计算机通常都会有数量不大的缓存。数组在内存里是连续存放的，所以，如果访问数组元素的时候能够按照顺序来，缓存可以起到极大的加速作用。

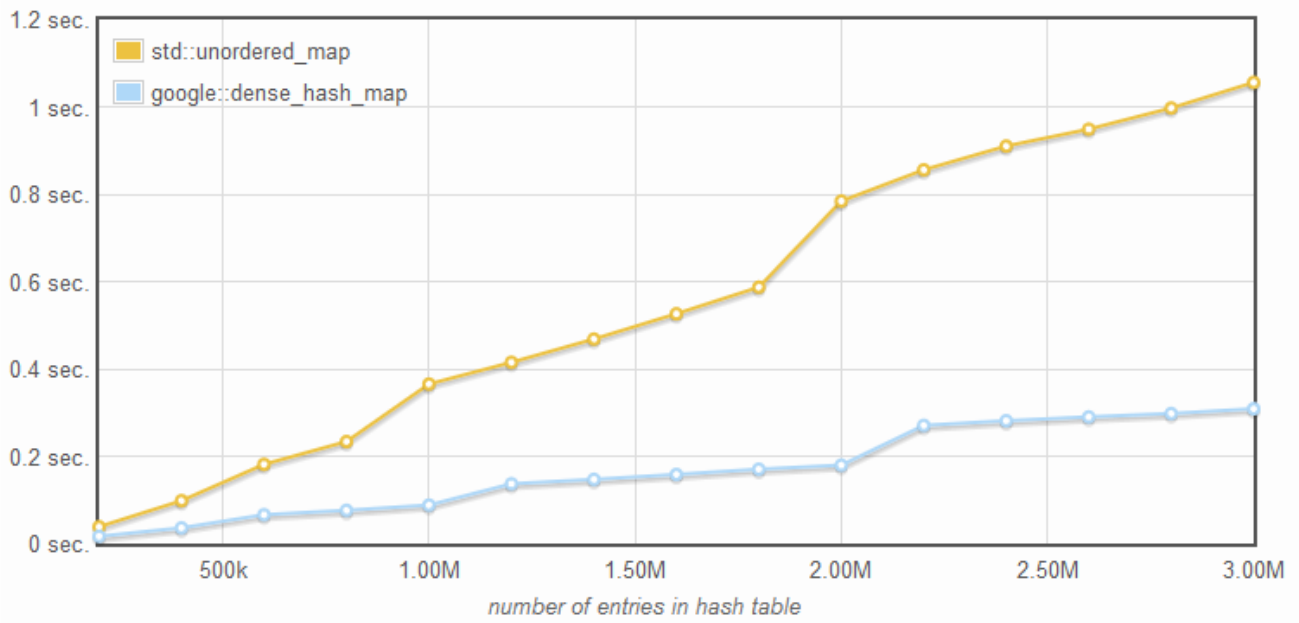
小王一开始的二重循环，恰恰没有有效地使用缓存，反而对数组元素类似随机访问。第二个版本就改正了这个错误，优化了性能。

举例 3：资深的程序员

小赵工作 4 年了，已经算是资深的 C++ 程序员，负责一个程序的开发和设计。他的一个程序需要使用一个 Map 的数据结构。他开始使用的是 STD 库的标准实现：`unordered_map`。但是他发现，在数据量大的时候，键值的插入操作需要的时间很长。虽然做了各种代码优化，但性能总是不尽人意。

其实，如果他了解 C++ 有些库有更高效的 Map 实现，比如 `google::dense_hash_map`，他或许可以酌情采用，从而大幅度提升性能。

很多的测试结果显示，`google::dense_hash_map`的性能可以比`std::unordered_map`快好几倍。下图（图片来自 https://tessil.github.io/ ）正是同一种测试环境下，两种实现的处理时间比较，我们可以清楚地看出性能的差距。



举例 4：技术专家

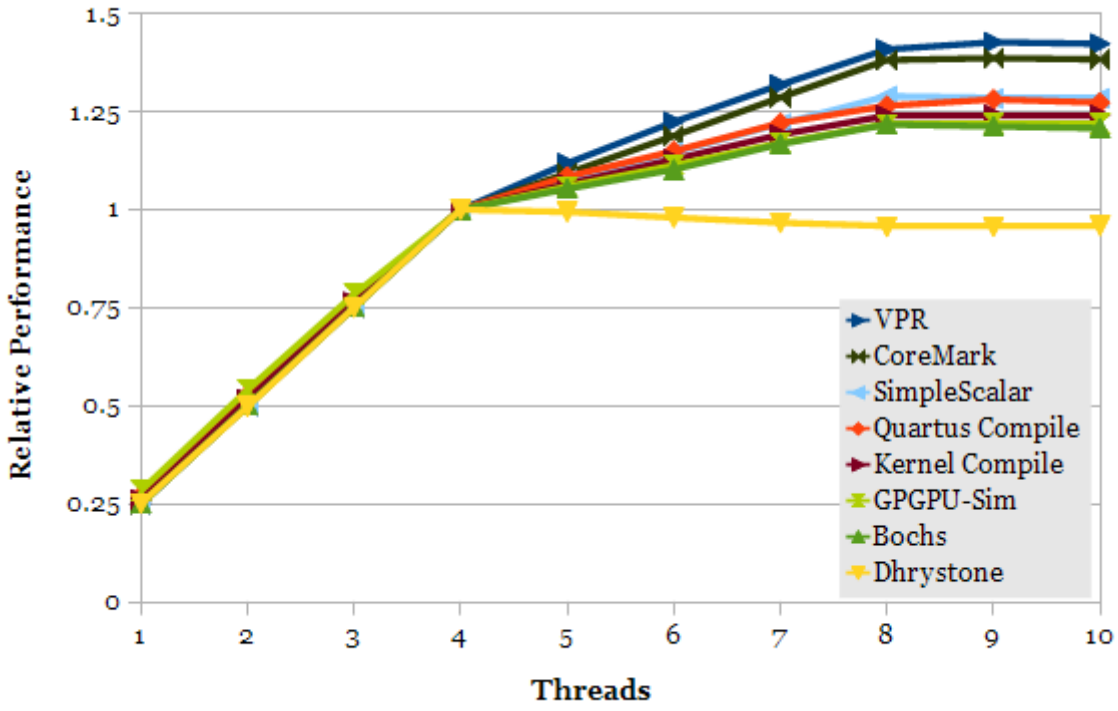
小刘工作 8 年了，在公司里已经算是不大不小的技术专家了。

有一天，他看到一份项目计划，其中有一段引起了他的兴趣。这份计划是为了提高服务器的 CPU 使用效率，提出把应用程序的线程池增大，建议程序线程池的主线程数目应该和服务器的逻辑 CPU 的数目相等。当然，这里的逻辑 CPU，就是我们通常说的虚拟内核数。

小刘这几年对硬件和操作系统钻研良多，他立刻指出，这样部署不妥，他建议降低主线程池大小到逻辑 CPU 的一半。技术讨论过程中，小刘给大家仔细讲解了原因，大家最后认可了他的建议，小刘也获得了大家的青睐。

小刘之所以这样建议，是因为他知道，服务器的逻辑 CPU 不是物理 CPU。在超线程技术（Hyper Threading）的情况下，服务器的吞吐量不是严格按照逻辑 CPU 的使用率来提升的，因为两个逻辑 CPU 其实共享很多物理资源。

比如下面这张图，就表示了在一台有 8 个逻辑 CPU 的服务器上，如果部署超过 4 个线程，得到的性能提升非常有限，甚至可能会带来其他不好的后果。这里具体的提升率和效果，取决于线程和应用程序的特性。（图片来自 http://blog.stuffedcow.net）



举例 5： 高级专家（架构师）

老周是公司里的架构师和高级专家。他最近对公司的一个重要业务进行了性能优化，用很小的代码改动，就给公司节省了几百万美元的运营成本（这是我身边发生的一个真实案例，除了名字不一样）。

这个业务的性能瓶颈是 CPU。因为业务量大，这个业务部署了 1 万台以上的服务器，占用了很大一部分数据中心的容量。

老周仔细研究了业务的逻辑，并且进行了性能测试和分析。他发现代码的执行过程卡在了 CPU 取指令的速度上：因为内存和缓存的物理特性，CPU 花了很大一部分时间在等待指令获取，从而造成了 CPU 浪费。

他经过考虑，决定进行**指令级别的提前获取优化**。具体来讲，就是用 GCC 的 `__builtin_prefetch`指令来预先提取关键指令，从而降低缓存的缺失比例，也就提高了 CPU 的使用效率。

下图是 GCC 关于这个指令的官方文档。

```
GCC Built-in Function:
    void __builtin_prefetch (const void *addr, ...)

This function is used to minimize cache-miss latency by moving data into a cache
before it is accessed. You can insert calls to __builtin_prefetch into code for
which you know addresses of data in memory that is likely to be accessed soon. If
the target supports them, data prefetch instructions will be generated. If the
prefetch is done early enough before the access then the data will be in the cache
by the time it is accessed.
```

经过这样的优化，一台服务器可以处理比以前多 50% 的请求，从而节省了相应比例的服务器和容量。从公司成本角度来看，这一优化节省了 3 千台以上的服务器，价值几百万美元，老周被 CEO 开会表扬，也是自然的事情了。

有趣的是，整个的代码改进只需要几行代码的改动，真真切切是“一字万金”。

总结

重要的事情需要多说几遍：每个 IT 从业人员，尤其是程序员，都需要关心代码性能。

如果不了解性能的知识，也许能写出可运行但性能不好的代码。但一个真正对工作、对公司和对自己负责的程序员一定会发现，性能不好的代码无异于耍流氓，不经用还隐患无穷，万万要不得。

换句话说，对程序员来说，生活不仅是眼前的代码，还有效率和性能的优化。唐代诗人孟郊在考中进士后写了一首《登科后》，其中有两句：“春风得意马蹄疾，一日看尽长安花。”

我们谁不希望写出来的代码也运行飞快，自己能春风得意呢？！

思考题

无论你工作几年了，也无论是现在具体做什么工作，你能举出一两个，因为代码性能不好并导致严重后果的例子吗？是什么样的性能问题呢？

换个角度来说，如果写代码的程序员一开始就考虑到各种性能问题，并且提前在代码里面解决，写出的代码跑得飞快，而且很稳定。这样靠谱的程序员你会不会给他点赞？

欢迎留言和我分享你的观点，也欢迎你把今天的内容分享给身边的朋友，和他一起讨论。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　开篇词 | 代码不要眼前的苟且，而要效率的提升和性能的优化

下一篇　02 | 程序员也要关心整个系统和公司成本吗?

精选留言 (14)

写留言

OlafOO
2019-12-02

老师对不同阶段程序员举的例子好经典

展开

作者回复: 😄，谢谢。祝愿你快速修炼到最高级!

6

Jxin
2019-12-03

有意思。文笔有优秀产品经理的雅致与风趣。逻辑有优秀程序员的清晰与精准。开篇两章，已见不凡。追了，追了。

展开

作者回复: 好高的评价！谢谢谢谢。

4

arsterc
2019-12-03

赞！ 以前都是零碎的学习性能方面的知识， 现在可以系统学习了. 期待!

2

权奥
2019-12-03

遇到过快速启动200个Python进程，触发了系统oom，导致一个重要服务被kill的问题

2

Bugzella
2019-12-03

最近在做一些后台性能提升的工作，一个主要方式就是使用redis一类的缓存，降低读取操作数据库的频率，可以提升应用的延迟，提高数据库的吞吐量。另外一个例子是数据库读取语调优对于大部分严重依赖数据库的应用很重要，希望在这里也能读到相关的心得和分享。

展开

2

旭东
2019-12-05

程序优化一般都在业务优化以后吧，业务流程的优化效果也不可忽视

1

Q
2019-12-04

第一，为什么那么多人不重视不了解性能优化，要怪就怪，现在的机器性能太强悍了，人类科技的进步，掩饰掉了大部分因为人为原因造成的性能问题，所以科技的惯性太隐蔽又太强悍。
第二，关键还在于业务数据量是否增长和公司是否重视性能测试，如果重视性能压测，那么早期都会发现系统性能的瓶颈和压力在哪里，否则总是会在生产过程某个时间点中出...

展开

作者回复: 体会的很好!

1 1

子龙
2019-12-02

没有遇到什么有意思的性能问题
感想，性能问题的解决，依赖人对整个计算机运行的理解
对程序运行环境的理解：什么语言、什么系统、什么CPU、什么硬盘等等
有可能出现哪些问题，重点观察对应的指标，印证猜想，找对应的方案

展开

作者回复: 很不错的感想。的确，知道得多很重要。

1

丁丁历险记
2019-12-05

例子一的程序员直接开除。
例子五的架构师鼓舞着我们前行（做死）

每天都在做死的路上

作者回复: 要允许年轻人犯错误，每个人都是从菜鸟到大虾的。😁

西西弗与卡夫卡
2019-12-03

早期为了快速发展业务，公司使用了Ruby，开发是快，但扛不住多少访问量。后来业务验证之后，就更换成Java。在不同业务阶段，在快速验证和提升性能之间选择不同策略。

作者回复: 对，Java应该更有效些。不过比起C++来，C++又胜一筹了。这就是开发速度和效率的关系。一般后端用C++和Java比较多，因为需要高效率。

:)
2019-12-03

老师讲得太好了！这课性价比 太高了！

展开

作者回复: 太谢谢了！更加有干劲了，努力做得更好！

humor
2019-12-03

以前在一个方法上使用了synchronous关键字，自己测试觉得没问题，发布到线上之后，系统就特别慢，各种超时...以后再也不敢随便用锁了

展开

作者回复: 恩，尽量把锁的范围变小；只锁绝对需要锁的地方。

夜空中最亮的星（华仔...
2019-12-02

老师讲的很好，写出跑的飞快的代码的程序员很让人佩服

作者回复: 非常感谢！

故事、自己写
2019-12-02

不求一字万金，一字千金即可～

展开

作者回复: 哈哈，你一定可以的！